Predicting Bicycle Rentals for a Bike Share Service in Washington D.C. and Surrounding Area

Prepared by : Jinping Bai

Krishna Kiruba

Leolein Paopchi

Paul Flemming

Sam Fawzi

Prepared for : Hashmat Rohian

CSDA1010 Basic Methods of Data Analytics - Winter 2020 Section III

Table of Contents

Abstract

The Washington D.C. bike share service was first in operation in 2008. It was the first such service in North America. One of the activities that the corporation that runs the service requires is the projected number of bike rentals on a per hour basis. This report addresses this activity by using historical data and test a number of Regression Models to determine the best algorithm for forecasting bike rentals on a per hour basis. A total of five scenarios were modeled and based upon the success criteria the Random Forest model achieved the required results. The following report discusses the methodology that was followed and the other models that were tested. This report includes a portion of the coding and a portion of the visualizations that were created. Attached with this report is a html file created with Jupyter Notebook which includes all the coding used for data understanding, feature engineering and modelling. This report plus the Jupyter Notebook file and the data can be found on Github at the following web address. https://github.com/Salampop/BIKE-SHARING-DATASET-Analysis

Introduction

"Life connected by pedal strokes." (Press Kit, 2020) This is the vision of Capital Bikeshare. A bike sharing service in Washington DC. A service that provides enjoyable and environmentally friendly transport from point A to point B. Washington DC was the first city in North America to offer a bike sharing service. This service started in August 2008 and started with 120 bikes and 10 stations. Since then the service has grown and spread into seven localities in and around Washington DC. The service now has 4300 bikes in 500 stations. This is a 50% growth year over year. (Press Kit, 2020)

One of the many challenges they have is the prediction of bike usage for future capital and operating budgeting requirements and for locations of future bike racks and bikes themselves. The algorithm developed for Capital Bikeshare will help in determining how the bike sharing service will expand or contract in the future.

The participants of the team involved in this study are Sam Fawzi, Jinping Bai, Krishna Kiruba, Leolein Paouchi, and Paul Flemming. The team brings together a wide variety of experience from the fields of business, finance, logistics, engineering and IT. This combined experience enables the team to analyze the project from different perspectives.

Background

The datasets (Bike Sharing Dataset Data Set) that have been provided for this study are usage statistics on a daily and hourly basis. Both files include fields about weather, working days, type of user (registered or not) and the total usage count per hour, in one file and usage per day in another file. The data we are working is from the years 2011 and 2012. One issue with working with this data set is that it is eight years old. To provide an algorithm that will predict usage for the rest of 2020 and into 2021, the existing dataset should be replaced with more recent data from 2018 and 2019. One aspect that this report does not take into account is the year over year increase in usage. To account for year over year increase a much larger dataset would need to be used that will encompass a larger number of years.

Objective

The objective of this study is to develop an algorithm to predict the number of system users per hour. This process will include looking at the variables and deciding if they are required in the modelling process. Looking for outliers in the data. Checking for missing data. Checking the independence of the variables to each other. A variety of models will initially be tested in determining the best way to develop the desired algorithm. The models that will be tested include Decision Tree, Random Forest, Train, test split model and Linear Regression model. A model with a score greater than 0.90 will be considered as a successful model.

All coding for data understanding, data preparation and modelling is provided in a separate html file that has been generated from a Jupyter Notebook using Python as the programming language. Select tables and graphs are included in this report.

Data Understanding

Data generated from Bike sharing systems makes it attractive for researchers as the duration of travel, departure location, time elapsed, weather and season are usually recorded. Datasets can be used for studying mobility in a city. The dataset (Bike Sharing Dataset Data Set) that we will be using has been retrieved from the University of California Irvine's Machine Learning Repository. The data compiles user information from the years 2011 and 2012 from Capital Bikeshare system, Washington DC. Data is aggregated on an hourly and daily basis. For our model, that will be developed, we will focus on data obtained on an hourly basis. Furthermore, weather and seasonal information alongside holiday schedule were obtained from the links below:

Weather Information: http://www.freemeteo.com
Holiday Schedule: http://dchr.dc.gov/page/holiday-schedule

The dataset (Bike Sharing Dataset Data Set) contains 17 attributes and 17,379 instances. Sixteen of the variables are listed in the table below in detail below:

Column Name (Feature)	**Column Description (Feature Description)**
instant	record index
dteday	date
season	season 1. Spring 2. Summer 3. Fall 4. Winter
yr	year (0: 2011, 1:2012)
mnth	month (1 to 12)
hr	hour (0 to 23)
holiday	weather day is holiday or not (extracted from http://dchr.dc.gov/page/holiday-schedule)
weekday	day of the week
workingday	if day is neither weekend nor holiday is 1, otherwise is 0.
weathersit	weather situation 1. Clear, Few clouds, Partly cloudy, Partly cloudy 2. Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist 3. Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds 4. Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
temp	Normalized temperature in Celsius. The values are derived via (t-t_min)/(t_max-t_min), t_min=-8, t_max=+39 (only in hourly scale)
atemp	Normalized feeling temperature in Celsius. The values are derived via (t-t_min)/(t_max-t_min), t_min=-16, t_max=+50 (only in hourly scale)
hum	Normalized humidity. The values are divided to 100 (max)
windspeed	Normalized wind speed. The values are divided to 67 (max)
casual	count of casual users
registered	count of registered users

Target Variable

We are trying to predict the count variable (cnt, the 17th variable of the dataset) i.e. count of total rentals including both casual and registered which will be behaving as a dependent variable for our analysis.

Preparation

To analyse the data, various Python libraries (i.e. pandas, numpy, matplotlib, seaborn) were used. Some of the analysis was also performed in R. Before starting the analysis, the Python packages were made available by running the following code.

```
Import pandas as pd
Import numpy as np
…
```

Before preparing the data for modeling we need to better understand our data, we will start by finding the correlations between our target and other features.

Importing and reviewing the dataset

The dataset was loaded using the following code:

```
hour=pd.read_csv('hour.csv')
```

Preview of data

The first 5 rows of the raw dataset are shown in the table below:

	instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
0	1	2011-01-01	1	0	1	0	0	6	0	1	0.24	0.2879	0.81	0.0	3	13	16
1	2	2011-01-01	1	0	1	1	0	6	0	1	0.22	0.2727	0.80	0.0	8	32	40
2	3	2011-01-01	1	0	1	2	0	6	0	1	0.22	0.2727	0.80	0.0	5	27	32
3	4	2011-01-01	1	0	1	3	0	6	0	1	0.24	0.2879	0.75	0.0	3	10	13
4	5	2011-01-01	1	0	1	4	0	6	0	1	0.24	0.2879	0.75	0.0	0	1	1

Summary of attributes

For each feature, the table below displays the Min, 1st and 3rd Quartiles, Median, Mean and Max

```
    instant           dteday          season            yr              mnth              hr           holiday
 Min.   :    1   2011-01-01:   24   Min.   :1.000   Min.   :0.0000   Min.   : 1.000   Min.   : 0.00   Min.   :0.00000
 1st Qu.: 4346   2011-01-08:   24   1st Qu.:2.000   1st Qu.:0.0000   1st Qu.: 4.000   1st Qu.: 6.00   1st Qu.:0.00000
 Median : 8690   2011-01-09:   24   Median :3.000   Median :1.0000   Median : 7.000   Median :12.00   Median :0.00000
 Mean   : 8690   2011-01-10:   24   Mean   :2.502   Mean   :0.5026   Mean   : 6.538   Mean   :11.55   Mean   :0.02877
 3rd Qu.:13034   2011-01-13:   24   3rd Qu.:3.000   3rd Qu.:1.0000   3rd Qu.:10.000   3rd Qu.:18.00   3rd Qu.:0.00000
 Max.   :17379   2011-01-15:   24   Max.   :4.000   Max.   :1.0000   Max.   :12.000   Max.   :23.00   Max.   :1.00000
                 (Other)   :17235
    weekday        workingday       weathersit         temp           atemp             hum           windspeed
 Min.   :0.000   Min.   :0.0000   Min.   :1.000   Min.   :0.020   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
 1st Qu.:1.000   1st Qu.:0.0000   1st Qu.:1.000   1st Qu.:0.340   1st Qu.:0.3333   1st Qu.:0.4800   1st Qu.:0.1045
 Median :3.000   Median :1.0000   Median :1.000   Median :0.500   Median :0.4848   Median :0.6300   Median :0.1940
 Mean   :3.004   Mean   :0.6827   Mean   :1.425   Mean   :0.497   Mean   :0.4758   Mean   :0.6272   Mean   :0.1901
 3rd Qu.:5.000   3rd Qu.:1.0000   3rd Qu.:2.000   3rd Qu.:0.660   3rd Qu.:0.6212   3rd Qu.:0.7800   3rd Qu.:0.2537
 Max.   :6.000   Max.   :1.0000   Max.   :4.000   Max.   :1.000   Max.   :1.0000   Max.   :1.0000   Max.   :0.8507

     casual         registered         cnt
 Min.   :  0.00   Min.   :  0.0   Min.   :  1.0
 1st Qu.:  4.00   1st Qu.: 34.0   1st Qu.: 40.0
 Median : 17.00   Median :115.0   Median :142.0
 Mean   : 35.68   Mean   :153.8   Mean   :189.5
 3rd Qu.: 48.00   3rd Qu.:220.0   3rd Qu.:281.0
 Max.   :367.00   Max.   :886.0   Max.   :977.0
```

Missing Data

The Python statement below was used to determine if the dataset has any missing values. There were no missing values found.

```
print(hour.isna().sum()/len(hour)*100)
```

```
instant       0.0
dteday        0.0
season        0.0
yr            0.0
mnth          0.0
hr            0.0
holiday       0.0
weekday       0.0
workingday    0.0
weathersit    0.0
temp          0.0
atemp         0.0
hum           0.0
windspeed     0.0
casual        0.0
registered    0.0
cnt           0.0
dtype: float64
```

Data Visualization

1- *Group the data set by hour and aggregate the count of trips to find the highest and lowest demand by hour.*

```
   sum  min  max       mean
hr
17  336860   15  976  461.452055
18  309772   23  977  425.510989
8   261001    5  839  359.011004
16  227748   11  783  311.983562
```

Code:

```
fig, ax = plt.subplots()
hr_cnt =
hour.groupby(['hr']).cnt.agg({'max','min','mean','sum'})
print (hr_cnt.sort_values(by=['sum'],ascending=False))
hr_cnt['sum'].plot()
ax.set_xlabel('Hour')
ax.set_ylabel('Count')
```

19	226789	11	743	311.523352
13	184919	11	760	253.661180
12	184414	3	776	253.315934
15	183149	7	750	251.233196
14	175652	12	750	240.949246
20	164550	11	567	226.030220
9	159438	14	426	219.309491
7	154171	1	596	212.064649
11	151320	10	663	208.143054
10	126257	8	539	173.668501
21	125445	6	584	172.314560
22	95612	9	502	131.335165
23	63941	2	256	87.831044
6	55132	1	213	76.044138
0	39130	2	283	53.898072
1	24164	1	168	33.375691
2	16352	1	132	22.869930
5	14261	1	66	19.889819
3	8174	1	79	11.727403
4	4428	1	28	6.352941

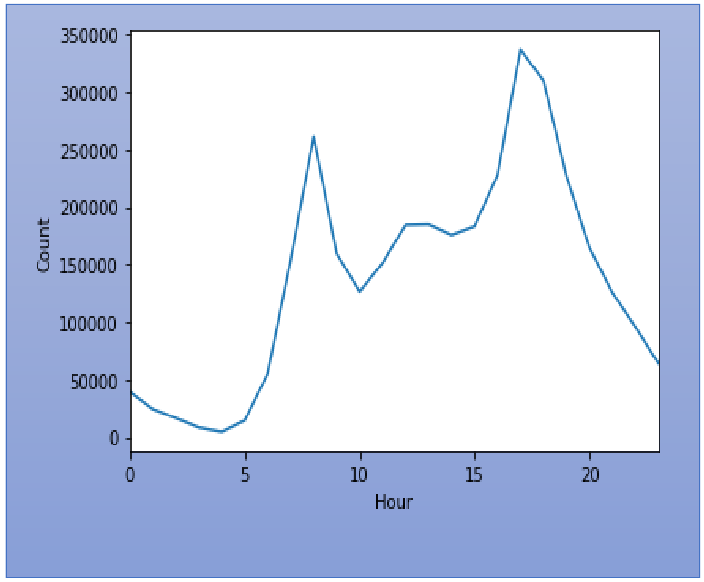


Figure 1

As we can see the demand is increasing during rush hours and decreases during late night and early morning.

2- Let’s do the same and find the highest and lowest demand by month

mnth	sum	min	max	mean
8	351194	1	941	238.097627
6	346342	1	900	240.515278
9	345991	1	977	240.773138
7	344948	1	913	231.819892
5	331686	1	873	222.907258
10	322352	1	963	222.158511
4	269094	1	822	187.260960
11	254831	1	729	177.335421
3	228920	1	957	155.410726
12	211036	1	759	142.303439
2	151352	1	610	112.865026
1	134933	1	559	94.424773

Code:

```
hr_mnth = hour.groupby(['mnth']).cnt.agg({'max','min','mean','sum'})

print (hr_mnth.sort_values(by=['sum'],ascending=False))

hr_mnth['sum'].plot()
```

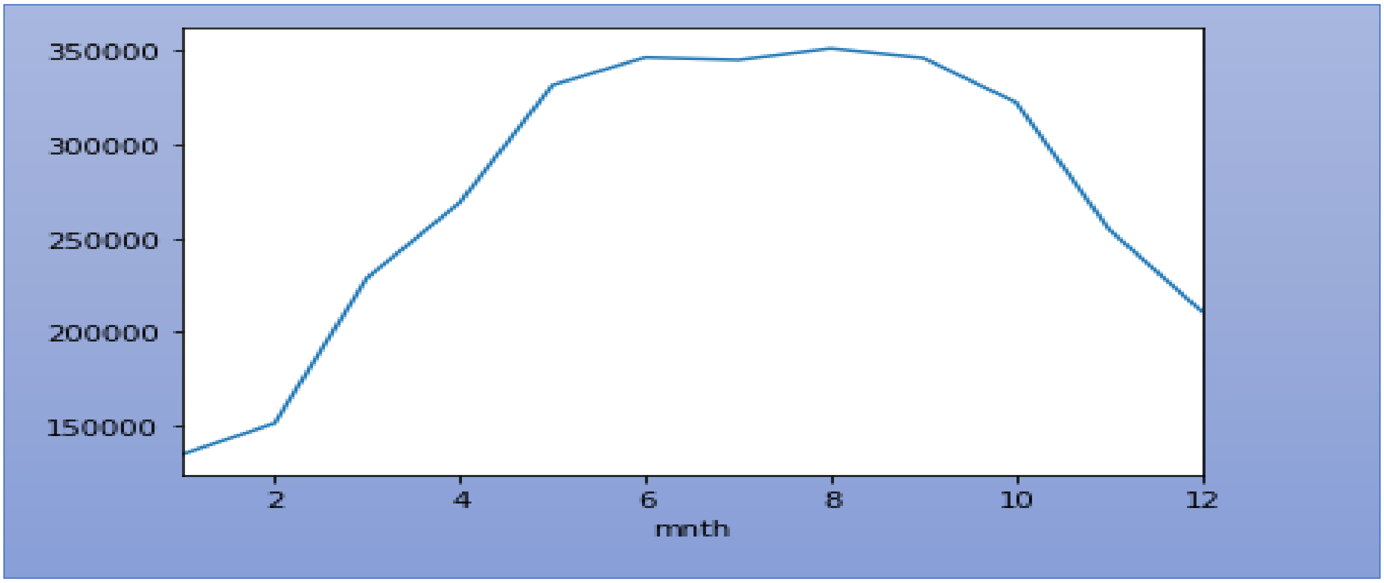


Figure 2

3- Correlation

corr = hour.corr()
print (corr)

```
              instant    season        yr      mnth        hr   holiday  \
instant      1.000000  0.404046  0.866014  0.489164 -0.004775  0.014723
season       0.404046  1.000000 -0.010742  0.830386 -0.006117 -0.009585
yr           0.866014 -0.010742  1.000000 -0.010473 -0.003867  0.006692
mnth         0.489164  0.830386 -0.010473  1.000000 -0.005772  0.018430
hr          -0.004775 -0.006117 -0.003867 -0.005772  1.000000  0.000479
holiday      0.014723 -0.009585  0.006692  0.018430  0.000479  1.000000
weekday      0.001357 -0.002335 -0.004485  0.010400 -0.003498 -0.102088
workingday  -0.003416  0.013743 -0.002196 -0.003477  0.002285 -0.252471
weathersit  -0.014198 -0.014524 -0.019157  0.005400 -0.020203 -0.017036
temp         0.136178  0.312025  0.040913  0.201691  0.137603 -0.027340
atemp        0.137615  0.319380  0.039222  0.208096  0.133750 -0.030973
hum          0.009577  0.150625 -0.083546  0.164411 -0.276498 -0.010588
windspeed   -0.074505 -0.149773 -0.008740 -0.135386  0.137252  0.003988
casual       0.158295  0.120206  0.142779  0.068457  0.301202  0.031564
registered   0.282046  0.174226  0.253684  0.122273  0.374141 -0.047345
cnt          0.278379  0.178056  0.250495  0.120638  0.394071 -0.030927

              weekday  workingday  weathersit      temp     atemp       hum
\
instant      0.001357   -0.003416   -0.014198  0.136178  0.137615  0.009577
season      -0.002335    0.013743   -0.014524  0.312025  0.319380  0.150625
yr          -0.004485   -0.002196   -0.019157  0.040913  0.039222 -0.083546
mnth         0.010400   -0.003477    0.005400  0.201691  0.208096  0.164411
hr          -0.003498    0.002285   -0.020203  0.137603  0.133750 -0.276498
holiday     -0.102088   -0.252471   -0.017036 -0.027340 -0.030973 -0.010588
weekday      1.000000    0.035955    0.003311 -0.001795 -0.008821 -0.037158
```

```
workingday  0.035955    1.000000    0.044672  0.055390  0.054667  0.015688
weathersit  0.003311    0.044672    1.000000 -0.102640 -0.105563  0.418130
temp       -0.001795    0.055390   -0.102640  1.000000  0.987672 -0.069881
atemp      -0.008821    0.054667   -0.105563  0.987672  1.000000 -0.051918
hum        -0.037158    0.015688    0.418130 -0.069881 -0.051918  1.000000
windspeed   0.011502   -0.011830    0.026226 -0.023125 -0.062336 -0.290105
casual      0.032721   -0.300942   -0.152628  0.459616  0.454080 -0.347028
registered  0.021578    0.134326   -0.120966  0.335361  0.332559 -0.273933
cnt         0.026900    0.030284   -0.142426  0.404772  0.400929 -0.322911

            windspeed    casual  registered       cnt
instant     -0.074505  0.158295    0.282046  0.278379
season      -0.149773  0.120206    0.174226  0.178056
yr          -0.008740  0.142779    0.253684  0.250495
mnth        -0.135386  0.068457    0.122273  0.120638
hr           0.137252  0.301202    0.374141  0.394071
holiday      0.003988  0.031564   -0.047345 -0.030927
weekday      0.011502  0.032721    0.021578  0.026900
workingday  -0.011830 -0.300942    0.134326  0.030284
weathersit   0.026226 -0.152628   -0.120966 -0.142426
temp        -0.023125  0.459616    0.335361  0.404772
atemp       -0.062336  0.454080    0.332559  0.400929
hum         -0.290105 -0.347028   -0.273933 -0.322911
windspeed    1.000000  0.090287    0.082321  0.093234
casual       0.090287  1.000000    0.506618  0.694564
registered   0.082321  0.506618    1.000000  0.972151
cnt          0.093234  0.694564    0.972151  1.000000
```

now let’s visualize the correlation

```
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
mask = np.triu(np.ones_like(corr, dtype=np.bool))
cmap = sns.diverging_palette(220, 10, as_cmap=True)
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5})
plt.title("Dataset feature' Correlation ", fontsize =10)
```

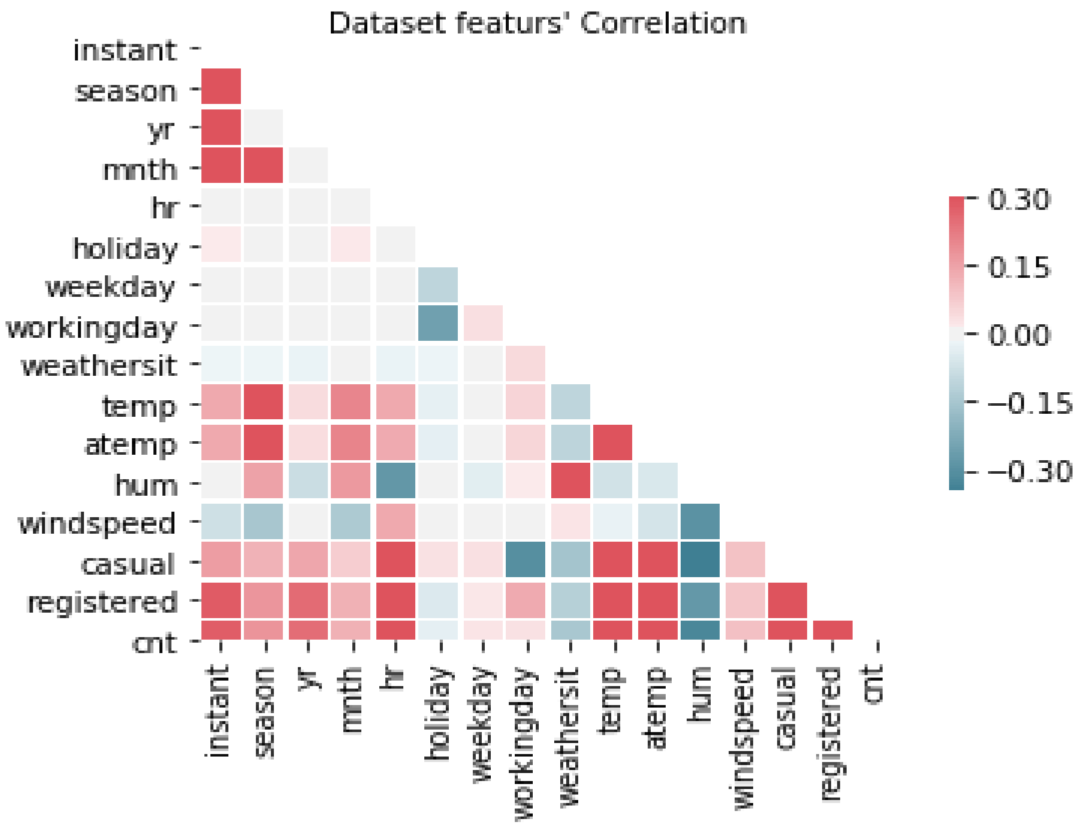


Figure 3

The figure above shows that there is a correlation between the count and (season, Month, temp, atemp, and hour)

4- Season distribution (counts by hour)

```
fig,ax = plt.subplots()
fig.set_size_inches(18, 8)
sns.pointplot(data=hour[['hr','cnt','season']],
              x='hr',
              y='cnt',
              hue='season',
              ax=ax)
ax.set(title="Season - hourly distribution of counts",xlabel='Hour',ylabel
='Total Count')
```

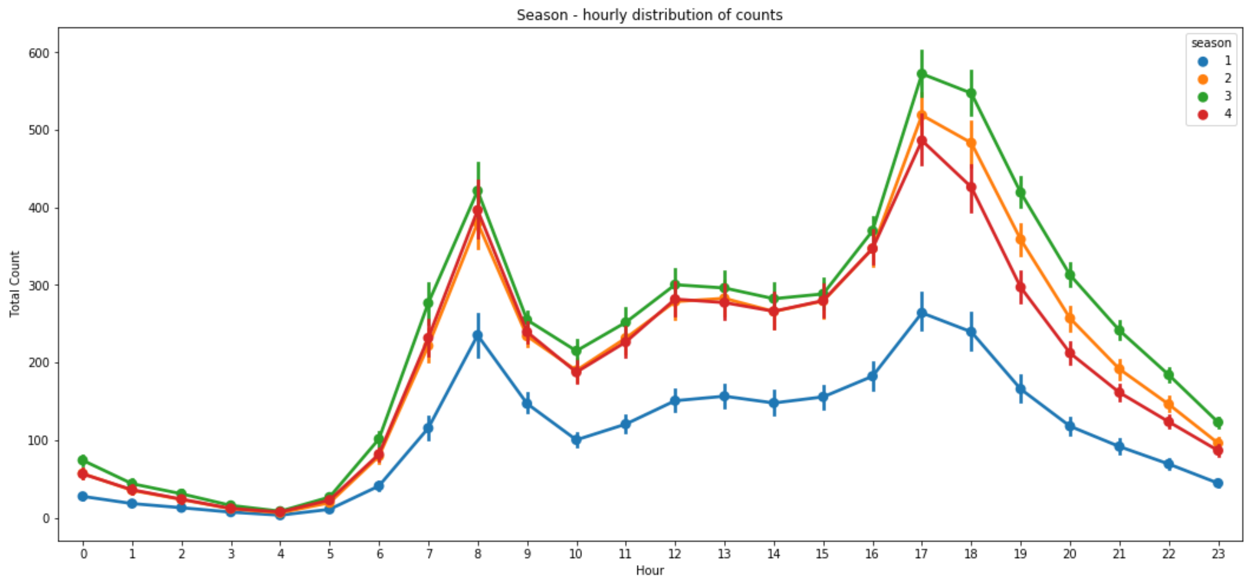


Figure 4

5- Hourly distribution by weekday

```
fig,ax = plt.subplots()
fig.set_size_inches(18, 8)
sns.pointplot(data=hour[['hr','cnt','weekday']],
              x='hr',y='cnt',hue='weekday',ax=ax)
ax.set(title="Weekday - hourly distribution of counts",xlabel='Hour',ylabe
l='Total Count')
```

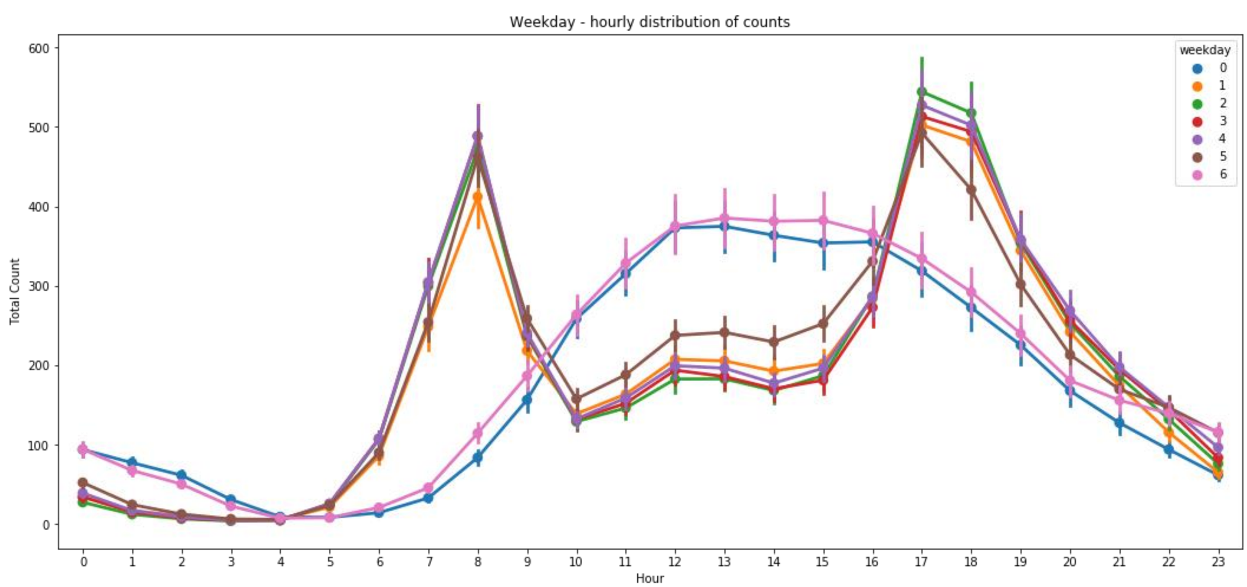


Figure 5

As we can see there is a higher hourly demand during the weekend between 10 am and 4 pm, and regular rush hours demands during the working days.

To further explore the dataset prior to modelling, we have switched to the R programming language, we can visualize how temperature values change over the four seasons. Prior to calculating the mean, standard deviation and median of the temperatures for each season, the values were denormalized to increase overall performance. The R code that was used is below:

```
#denormalize temp values
tconvert <- function(min, max, vector){
  result <- vector * (max - min) + min
  return (result)
}
#apply function and denormalize
BikeShare$temp <- tconvert(-8, 39, BikeShare$temp)
BikeShare$atemp <- tconvert(-16, 50, BikeShare$atemp)

#calculate mean by each season for temp

BikeShare.agg <- BikeShare %>%
  group_by(season) %>%
  summarise(
    temp.min = min(temp),
    temp.max = max(temp),
    temp.med = median(temp),
    temp.stdev = sd(temp),
    temp.mean = mean(temp),
    count = n())
BikeShare.agg
```

Mean, median, Max, Min, Median and standard deviation of temperature by season is shown below:

A tibble: 4 x 7

	season	temp.min	temp.max	temp.med	temp.stdev	temp.mean	count
	<chr>	*<dbl>*	*<dbl>*	*<dbl>*	*<dbl>*	*<dbl>*	*<int>*
1	fall	9.86	39	24.9	4.41	25.2	4496
2	spring	-7.06	25.8	5.16	5.58	6.06	4242
3	summer	-0.480	36.2	18.3	6.54	17.6	4409
4	winter	-1.42	27.7	11.7	5.74	11.9	4232

We can create a boxplot for temperature by season. R code is displayed below:

```
boxplot(temp ~ season,
        data = BikeShare,
        xlab = "Season",
        ylab = "Temperature",
        main = "Temperature by Season",
        col = "orange")
```

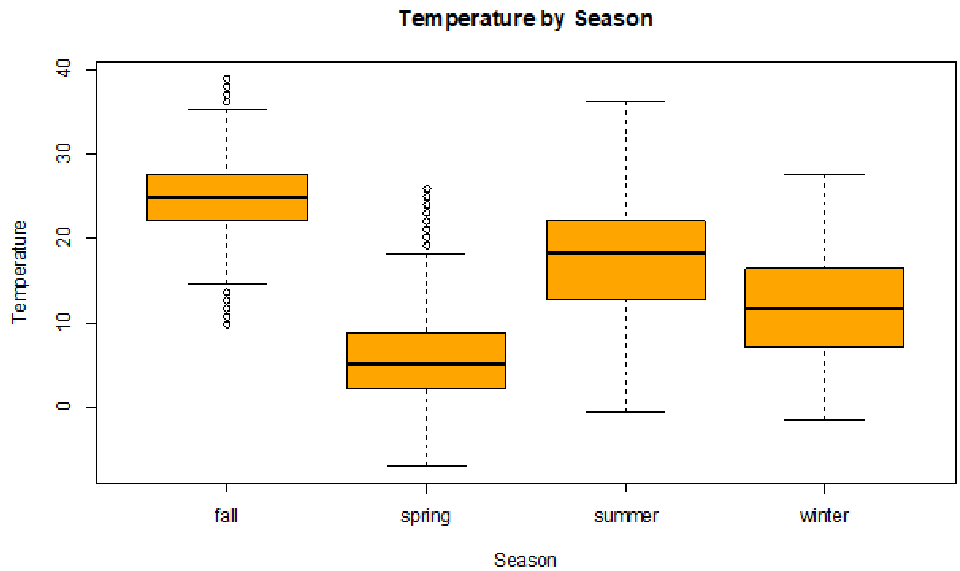


Figure 6

We can visualize the weather condition to the number of total bike rentals and compare the highest and the lowest counts. A bean plot was created to display the number of bike rentals vs weather condition. R code is shown below:

```r
install.packages("beanplot")
library("beanplot")
require("beanplot")
require("RColorBrewer")
bean.cols <- lapply(brewer.pal(6, "Set3"),
                    function(x){return(c(x, "black", "gray", "red"))})
beanplot(cnt ~ weathersit,
         data = BikeShare,
         main = "Bike Rents by Weather Condition",
         xlab = "Weather Condition",
         ylab = "# of Rentals",
         col = bean.cols,
         lwd = 1,
         what = c (1,1,1,0),
         log = ""
)
```

From the beanplot, we can determine that the lowest # of rentals occurs when weather type = 4 (Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog), while the highest mean value of bike rentals occurred when weather type = 1 (clear, partly cloudy)

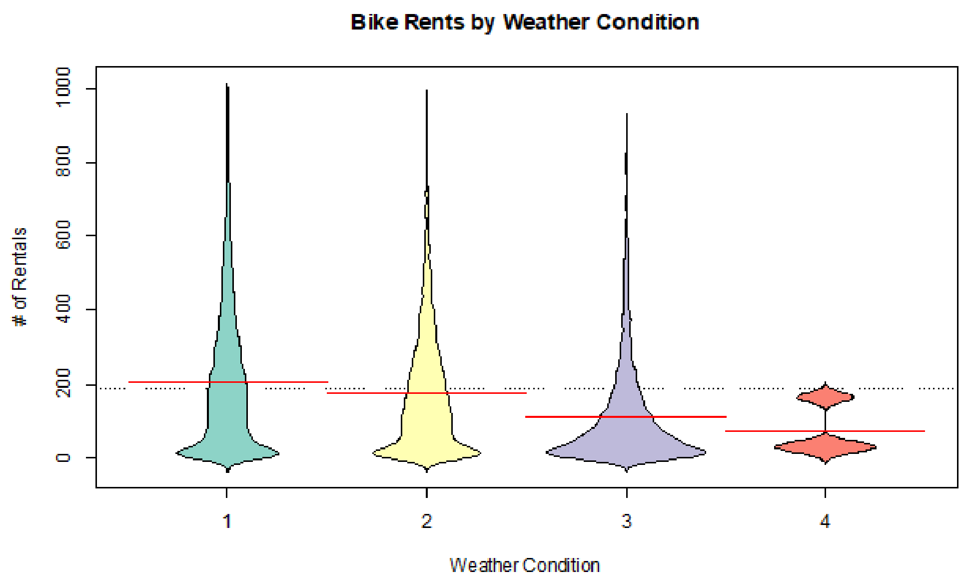


Figure 7

The line graph created below represents the total ridership per month for working and non-working days Based on the graph for all working days in the year, ridership was the highest during August. (Workingday =1). For weekends and holidays ridership was high in September. R code used is below

```
mnth_count<- BikeShare %>%
  select(mnth,workingday,cnt)
df <- mnth_count%>%
  group_by(mnth,workingday) %>%
  summarise(cnt = sum(cnt))
df$mnth <- as.factor(df$mnth)
df$workingday <- as.character(df$workingday)
point <- format_format(big.mark = ",", scientific = FALSE)
ggplot(df, aes(mnth,cnt)) + labs(title="Monthly ridership based on working and holiday")+
  geom_line(aes(color=workingday, group=workingday))+ scale_y_continuous(labels = point)
```

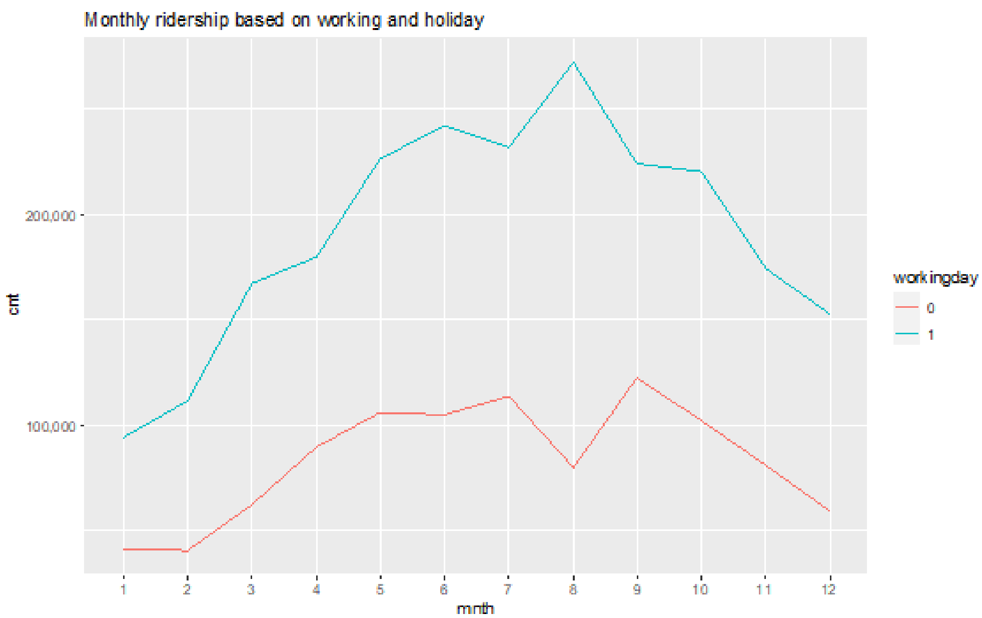


Figure 8

The rest of the code and visualizations in the report have been produced using Python.

Now let's have a look at the linear relationship between the temperature, season and the rides count.

sns.lmplot('temp','cnt',row='workingday',col='season',data=day,palette='RdBu_r',fit_reg=True)

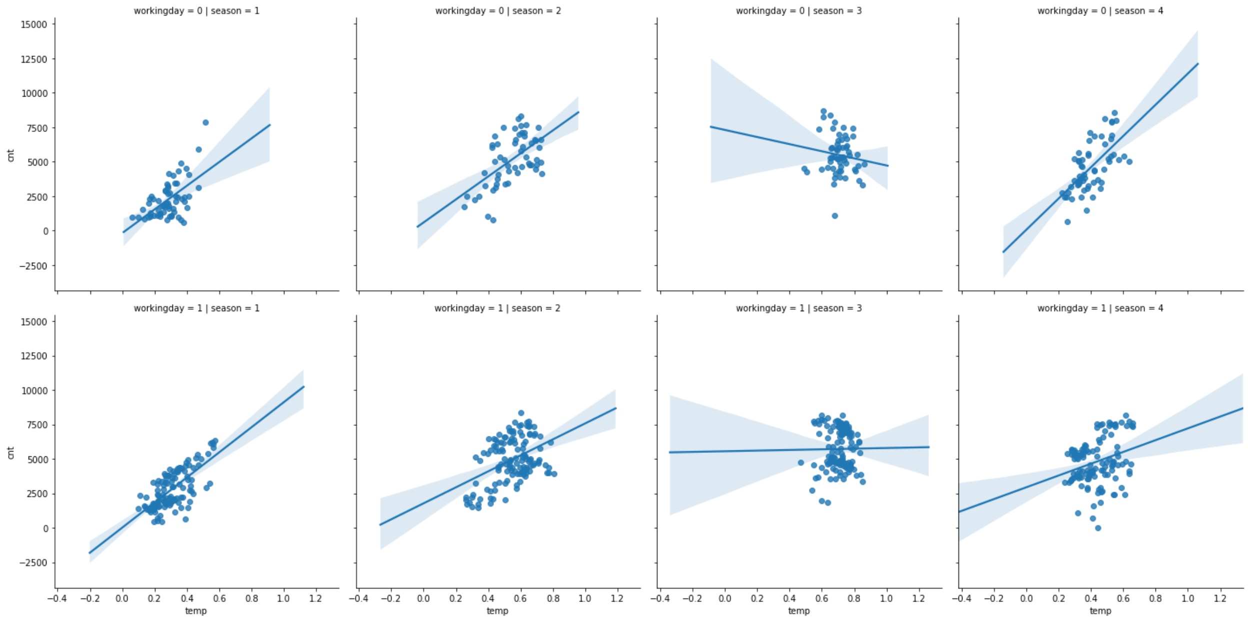


Figure 9

now let’s see the demand per hour during different seasons and compare the results between the working and non working days

```
G1= hour

G1['workingday'] = np.where(G1['workingday'] == '0', 'Not Working Day', G1['workingday'])

G1['workingday'] = np.where(G1['workingday'] =='1', 'Working Day', G1['workingday'])

g = sns.catplot(x="hr", y="cnt", hue="workingday", col="season",  data=G1, kind="bar",
height=10, aspect=1)
```

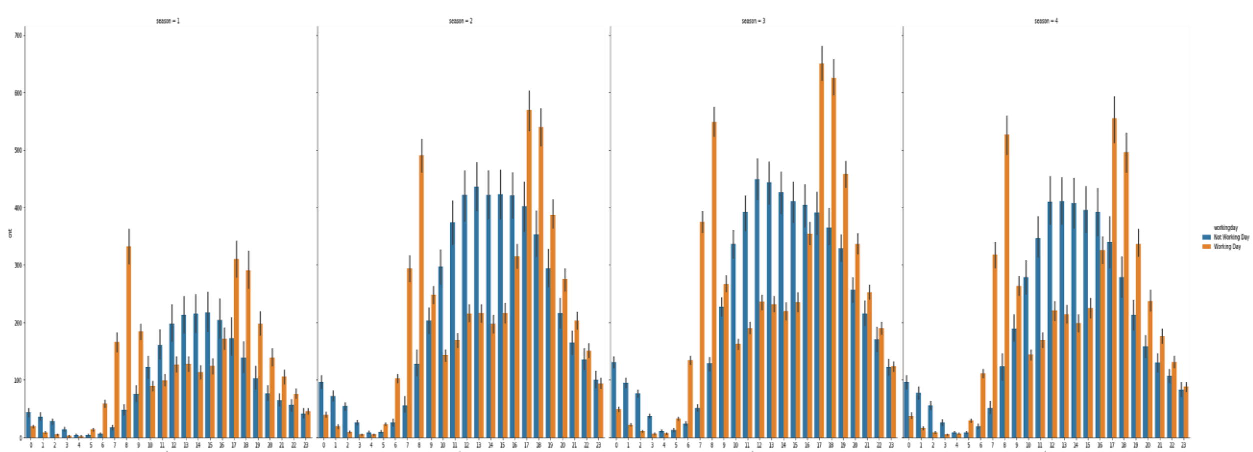
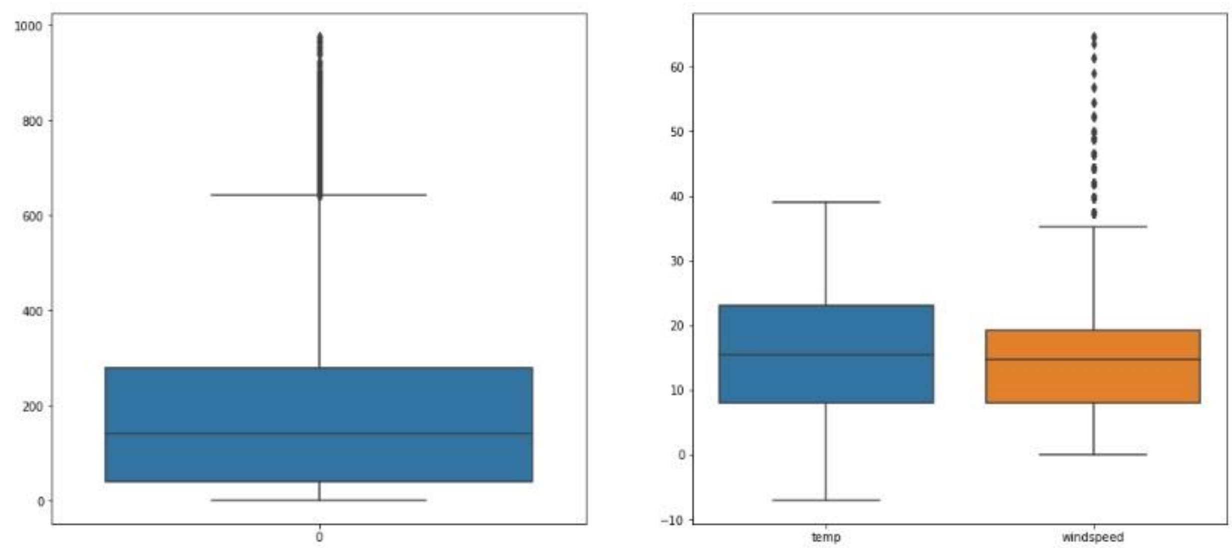


Figure 10

Outliers

An outlier is an observation that is unlike the other observations. It is rare, or distinct, or does not fit in some way. Let's have a look at our data set using boxplot.

```
fig,(ax1,ax2) = plt.subplots(ncols=2)
fig.set_size_inches(18, 8)
sns.boxplot(data=hour['cnt'],ax=ax1)
sns.boxplot(data=hour[['temp','windspeed']],ax=ax2)
```



For the outlier in the number of rides we can assume that there was a high demand in a couple of days and the reason could be a long holiday, special event, or something else. the temperature looks great! Yes because the data set was clean and the Temperature is normalized The values are derived via (t-t_min)/(t_max-t_min), t_min=-8, t_max=+39, now the wind speed, we can see there was a couple of times were the wind speed was not normal.

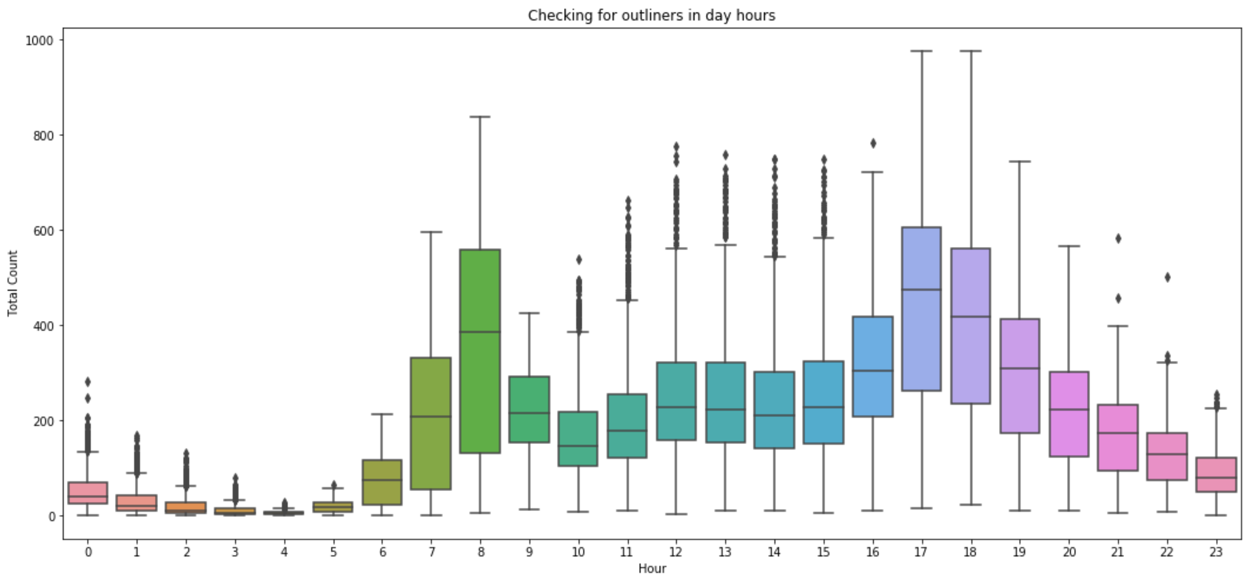
Now let’s look at the hourly demand using boxplot.

fig,ax = plt.subplots()

fig.set_size_inches(18, 8)

sns.boxplot(data=hour[['cnt', 'hr']],x='hr',y='cnt',ax=ax)

ax.set(title="Checking for outliers in day hours",xlabel='Hour',ylabel='Total Count')


Checking for outliners in day hours
Total Count
0
200
400
600
800
1000
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
Hour

Feature Engineering and Data Transformation

Based on the above analysis and visualizations we found that the features that affect the cnt target variable are (Hour, Season, Temperature, weather the day is working day or not, and the weather).

From the above we have 2 categorical columns (Season and weather). And we will transform the hours as well to eliminate the numerical value bias.

```
In [453]:  MOD_READY = hour

In [454]:  MOD_READY.columns

Out[454]: Index(['instant', 'dteday', 'season', 'yr', 'mnth', 'hr', 'holiday', 'weekday',
                 'workingday', 'weathersit', 'temp', 'atemp', 'hum', 'windspeed',
                 'casual', 'registered', 'cnt'],
                dtype='object')

In [455]:  MOD_READY.drop(['dteday','weekday','atemp','registered','casual','hum','windspeed','yr','instant','holiday'], axis=1, inplace

In [456]:  MOD_READY = pd.get_dummies(MOD_READY, columns=['hr','season','workingday','weathersit'],drop_first=False)
```

A close look to our dataset after transformation

```
In [458]:  MOD_READY
```

	mnth	temp	cnt	hr_0	hr_1	hr_2	hr_3	hr_4	hr_5	hr_6	...	season_1	season_2	season_3	season_4	workingday_0	workingday_1	weathersit_1	weathersit_2
0	1	0.24	16	1	0	0	0	0	0	0	...	1	0	0	0	1	0	1	0
1	1	0.22	40	0	1	0	0	0	0	0	...	1	0	0	0	1	0	1	0
2	1	0.22	32	0	0	1	0	0	0	0	...	1	0	0	0	1	0	1	0
3	1	0.24	13	0	0	0	1	0	0	0	...	1	0	0	0	1	0	1	0
4	1	0.24	1	0	0	0	0	1	0	0	...	1	0	0	0	1	0	1	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
17374	12	0.26	119	0	0	0	0	0	0	0	...	1	0	0	0	0	1	0	1
17375	12	0.26	89	0	0	0	0	0	0	0	...	1	0	0	0	0	1	0	1
17376	12	0.26	90	0	0	0	0	0	0	0	...	1	0	0	0	0	1	1	0
17377	12	0.26	61	0	0	0	0	0	0	0	...	1	0	0	0	0	1	1	0
17378	12	0.26	49	0	0	0	0	0	0	0	...	1	0	0	0	0	1	1	0

Let us take a look at the columns of the data after the transformation

Data columns (total 37 columns):
mnth 17379 non-null int64
temp 17379 non-null float64
cnt 17379 non-null int64
hr_0 17379 non-null uint8
hr_1 17379 non-null uint8
hr_2 17379 non-null uint8
hr_3 17379 non-null uint8
hr_4 17379 non-null uint8
hr_5 17379 non-null uint8
hr_6 17379 non-null uint8
hr_7 17379 non-null uint8
hr_8 17379 non-null uint8
hr_9 17379 non-null uint8

```
hr_10        17379 non-null uint8
hr_11        17379 non-null uint8
hr_12        17379 non-null uint8
hr_13        17379 non-null uint8
hr_14        17379 non-null uint8
hr_15        17379 non-null uint8
hr_16        17379 non-null uint8
hr_17        17379 non-null uint8
hr_18        17379 non-null uint8
hr_19        17379 non-null uint8
hr_20        17379 non-null uint8
hr_21        17379 non-null uint8
hr_22        17379 non-null uint8
hr_23        17379 non-null uint8
season_1     17379 non-null uint8
season_2     17379 non-null uint8
season_3     17379 non-null uint8
season_4     17379 non-null uint8
workingday_0   17379 non-null uint8
workingday_1   17379 non-null uint8
weathersit_1   17379 non-null uint8
weathersit_2   17379 non-null uint8
weathersit_3   17379 non-null uint8
weathersit_4   17379 non-null uint8
dtypes: float64(1), int64(2), uint8(34)
```

Modelling

Ordinary Least Squares Linear Regression Model

The first model that will be calculated will use an Ordinary Least Squares Linear Regression model. The code to run this model in Python is presented below.

```
### STATSMODELS ###

import pandas as pd

import seaborn as sns

import statsmodels.formula.api as smf

from sklearn.linear_model import LinearRegression

from sklearn import metrics

from sklearn.model_selection  import train_test_split

import numpy as np
```

create a fitted model

```
ML = smf.ols(formula='cnt ~ hr_0 + hr_1 +hr_2+ hr_3+ hr_4 +hr_5 +hr_6+ hr_7+
        hr_8+ hr_9+ hr_10+ hr_11+ hr_12+ hr_13+ hr_14+ hr_15+ hr_16+ hr_17
        +hr_18 +hr_19+ hr_20+ hr_21+ hr_22+ hr_23 + season_1 + season_2 + season_3 +
        season_4 + workingday_0 + workingday_1 + weathersit_1 + weathersit_2 +
        weathersit_3+ weathersit_4 + temp + atemp + hum + windspeed + yr_0 +yr_1+
        mnth_1+ mnth_2+ mnth_3+ mnth_4+ mnth_5+ mnth_6+ mnth_7 +mnth_8+ mnth_9+ mnth_10+
        mnth_11+ mnth_12+ SUN_0 +SUN_1', data=MOD_READY).fit()
```

The following code prints a summary of the calculations performed by the model.

```
print(ML.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    cnt   R-squared:                       0.685
Model:                            OLS   Adj. R-squared:                  0.684
Method:                 Least Squares   F-statistic:                     803.0
Date:                Thu, 23 Apr 2020   Prob (F-statistic):               0.00
Time:                        12:14:38   Log-Likelihood:            -1.0499e+05
No. Observations:               17379   AIC:                         2.101e+05
Df Residuals:                   17331   BIC:                         2.105e+05
Df Model:                          47
Covariance Type:            nonrobust
================================================================================
                   coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------
Intercept       49.2776      5.074      9.711      0.000      39.332      59.224
hr_0          -119.8297      3.873    -30.938      0.000    -127.422    -112.238
hr_1          -137.1126      3.890    -35.248      0.000    -144.737    -129.488
hr_2          -146.1438      3.929    -37.193      0.000    -153.846    -138.442
hr_3          -156.8199      3.988    -39.321      0.000    -164.637    -149.003
hr_4          -160.0424      4.007    -39.939      0.000    -167.897    -152.188
hr_5          -144.5786      3.868    -37.381      0.000    -152.160    -136.998
hr_6           -87.4308      3.818    -22.898      0.000     -94.915     -79.947
hr_7            45.8729      3.882     11.816      0.000      38.263      53.482
hr_8           186.2301      3.852     48.343      0.000     178.679     193.781
hr_9            38.4699      3.829     10.047      0.000      30.965      45.975
hr_10          -16.2507      3.829     -4.244      0.000     -23.756      -8.745
hr_11            9.0873      3.856      2.357      0.018       1.530      16.645
hr_12           48.3274      3.895     12.408      0.000      40.693      55.962
hr_13           43.2560      3.931     11.003      0.000      35.550      50.962
hr_14           27.3895      3.963      6.911      0.000      19.622      35.158
hr_15           36.8410      3.975      9.269      0.000      29.050      44.632
hr_16           98.9689      3.959     24.996      0.000      91.208     106.730
hr_17          253.9379      3.829     66.312      0.000     246.432     261.444
hr_18          223.7352      3.763     59.455      0.000     216.359     231.111
hr_19          116.9006      3.839     30.454      0.000     109.377     124.425
hr_20           37.3178      3.830      9.744      0.000      29.811      44.824
hr_21          -12.0850      3.831     -3.154      0.002     -19.595      -4.575
hr_22          -48.9911      3.840    -12.758      0.000     -56.518     -41.464
hr_23          -87.7724      3.852    -22.788      0.000     -95.322     -80.223
season_1       -22.6152      3.362     -6.727      0.000     -29.205     -16.025
season_2        15.5991      3.402      4.585      0.000       8.931      22.268
```

```
season_3          9.8273      3.552      2.767      0.006       2.866      16.789
season_4         46.4664      3.483     13.340      0.000      39.639      53.294
workingday_0     21.1453      2.681      7.886      0.000      15.890      26.401
workingday_1     28.1323      2.661     10.572      0.000      22.916      33.348
weathersit_1     45.5315     13.618      3.343      0.001      18.839      72.224
weathersit_2     35.4109     13.627      2.599      0.009       8.700      62.122
weathersit_3    -18.9661     13.744     -1.380      0.168     -45.905       7.973
weathersit_4    -12.6987     45.403     -0.280      0.780    -101.694      76.297
temp              2.5471      0.628      4.058      0.000       1.317       3.777
atemp             1.8599      0.464      4.012      0.000       0.951       2.769
hum              -0.8492      0.056    -15.270      0.000      -0.958      -0.740
windspeed        -0.3886      0.093     -4.183      0.000      -0.571      -0.207
yr_0            -18.0154      2.665     -6.760      0.000     -23.239     -12.791
yr_1             67.2930      2.645     25.442      0.000      62.109      72.477
mnth_1           -3.7598      4.578     -0.821      0.411     -12.732       5.213
mnth_2            0.0568      4.405      0.013      0.990      -8.578       8.692
mnth_3           11.4384      3.437      3.328      0.001       4.701      18.176
mnth_4            2.9202      4.120      0.709      0.478      -5.154      10.995
mnth_5           20.9503      4.352      4.814      0.000      12.420      29.480
mnth_6            7.4150      3.983      1.862      0.063      -0.392      15.222
mnth_7          -13.1131      4.757     -2.756      0.006     -22.438      -3.788
mnth_8            7.0624      4.532      1.558      0.119      -1.821      15.946
mnth_9           28.4959      3.711      7.680      0.000      21.223      35.769
mnth_10          11.7102      4.213      2.779      0.005       3.452      19.969
mnth_11         -14.4734      4.360     -3.319      0.001     -23.020      -5.927
mnth_12          -9.4252      3.743     -2.518      0.012     -16.761      -2.089
SUN_0            21.1966      2.856      7.421      0.000      15.598      26.795
SUN_1            28.0810      2.930      9.583      0.000      22.337      33.825
==============================================================================
Omnibus:                     1111.512   Durbin-Watson:                   0.503
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             2239.585
Skew:                           0.448   Prob(JB):                         0.00
Kurtosis:                       4.514   Cond. No.                     7.50e+16
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The smallest eigenvalue is 1.52e-26. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

Train Test Split Linear Regression Model – Feature Engineering Applied and Denormalized Data

The next model that was tested with the data is train test split linear regression model. The Python code to run this model is shown below.

```
# fit a model
from sklearn.metrics import mean_squared_error
from math import sqrt
lm = linear_model.LinearRegression()
model_FIT = lm.fit(X_train, y_train)
predictions = lm.predict(X_test)
```

and the code to display the results;

```
print ('Score:', model_FIT.score(X_test, y_test))

y_pred = lm.predict(X_test)

print('RMSE: %.2f' % sqrt(mean_squared_error(y_test, y_pred)))
```

And the results;

```
Score: 0.6825517917436095

RMSE: 103.35
```

Train Test Split Linear Regression Model – No Feature Engineering Applied and Denormalized Data

The same model was used again, but with data that had no feature engineering applied to the data. The results calculated from that model are as follows;

```
Score: 0.18828258688123697

RMSE: 164.36
```

Train Test Split Linear Regression Model – Feature Engineering Applied and Normalized Data

The train test split linear regression model was used again with data that has been normalized. The results of this model are shown below;

```
Score: 0.40694214000650825

RMSE: 141.00
```

Random Forest Model – Feature Engineering Applied and Denormalized Data

Random Forest method was selected for the next model run. The code for Random Forest is shown below and the results follow.

```
from sklearn.ensemble import RandomForestRegressor


regressor_RF = RandomForestRegressor(n_estimators=20, random_state=0)

regressor_RF.fit(X_train_R, y_train_R)
```

y_pred_R = regressor_RF.predict(X_test_R)

Results:

```
Mean Absolute Error: 33.432106382541505
Mean Squared Error: 2925.710200523883
Root Mean Squared Error: 54.08983453962383
0.9123227872539165
```

Decision Tree Classification

Decision Tree Classification is the last model that will be run. The Python code for this model is as follows:

```
X_DR = MOD_READY[feature_cols]
y_DR = MOD_READY.cnt
# create training and testing vars
X_train_DR, X_test_DR, y_train_DR, y_test_DR = train_test_split(X_DR, y_DR, test_size=0.2)
#3 Fitting the Decision Tree Regression Model to the dataset
# Create the Decision Tree regressor object here
from sklearn.tree import DecisionTreeRegressor
#DecisionTreeRegressor class has many parameters. Input only #random_state=0 or 42.
regressor_DR = DecisionTreeRegressor(random_state=0)
#Fit the regressor object to the dataset.
regressor_DR.fit(X_train_DR,y_train_DR)
```

This model produced a score of:

```
0.8421257752664962
```

Model Summary

The code below is used to produce a summary table of the modelling results.

Model_Scores = pd.DataFrame([['Linear regression with Normalized hours model', scaled_Hours_Model_Score,scaled_Hours_Model_RMSE],['Linear Reg with Train test split Model',FIT_Model_Score,FIT_Hours_Model_RMSE],

['Model Without feature engineering',Model_NO_FEATURE_Score,Hours_NO_FEATURE_SModel_RMSE],['Decision Tree Classifications',regressor_DR_SCORE,regressor_DR_SCORE_RMSE],['Random Forest'

,regressor_RF_SCORE,regressor_RF_SCORE_RMSE]], columns=['Model', 'Score', 'RMSE'])

print(Model_Scores)

And here are the results;

```
     Model                                         Score      RMSE
0 Linear regression with Normalized hours model   0.406942 141.001664
1 Linear Reg with Train test split Model          0.682552 103.350358
2 Model Without feature engineering               0.188283 164.359557
3 Decision Tree Classifications                   0.842126 72.584501
4 Random Forest                                   0.912323 54.089835
```

Summary

After a thorough investigation of the data provided and a series of model runs, the Random Tree model provided the most accurate result. There is a limitation of the algorithm that was produced. That limitation does not account for growth of the system over many years. Before this algorithm is used in a production setting the data will need to be updated. The data used was from the years 2011 and 2012. To create an algorithm for the current year, usage data and weather data would need to be gathered for the years from 2013 to 2019. Using all data that is available will allow us to create a model that will also be able to take into account yearly and monthly changes in bike rental usage. This new algorithm will allow greater accuracy in predictions further into the future.

Once the algorithm is accepted by Bikeshare and deployed in a production environment the results of the model should be compared to the actual usage data as it becomes available.   Periodically the algorithm should be updated. This updating process would involve combined new data collected since the last update and combining the new data set with the old data set. By constantly updating the algorithm, the accuracy of future bike rental predictions will become closer to reality.

Overall, this project was a success. First, we produced an algorithm that is able to achieve the business success criteria score of 0.9, our modelled score is 0.912. Second, we were also able to see were limitations were in the data set that was used to create the algorithm. Third, we have provided recommendations on how the algorithm can be improved moving into the future.

References

Bike Sharing Dataset Data Set. (n.d.). Retrieved from UCI Machine Learning Repository: http://archive.ics.uci.edu/ml/datasets/bike+sharing+dataset

Press Kit. (2020, 04 23). Retrieved from Capital Bikeshare: https://capitalbikeshare.com/press-kit